Intro JS

INFORMACIÓN DE CONTACTO DEL INSTRUCTOR

Kevin Martínez - https://www.linkedin.com/in/kevinjmartinez/ - Discord Tag: kevinccbsg#2306

Ejercicios y Wimblecode

Consideraciones generales

- **No se permite** el uso de librerías. Todo el código tiene que ser creado por el alumno.
- No es necesario crear un html para cada solución.

Ejercicio 1

Crea un archivo **ejercicio1.js** que tenga un objeto usuario con los siguientes campos:

- Nombre (el tuyo o inventado)
- Apellidos (el tuyo o inventado)
- Una lista con los temas del bootcamp Node.js, Git y react con sus nombres y fechas de inicio de cada módulo. Fecha en formato “YYYY-MM-DD”
- Si estás en busqueda activa con un valor de verdadero o false

En este archivo queremos mostrar por pantalla la fecha de inicio del módulo de react del objeto que hemos creado anteriormente.

Ejercicio 2 Arreglar bug

Nuestro cliente está intenando calcular el promedio de una lista de números pero nos dice que no funciona. No nos da el error, solo este código que es el que tiene en producción. Para este ejercicio tenemos que crear un archivo llamado **bug.js** con la solución.

```
const calcularPromedio = (numeros) => {
  let sumaTotal = 0;

  for (let i = 0; i <= numeros.length; i++) {
    sumaTotal += numeros[i];
  }

  const promedio = sumaTotal / numeros.length;
  return promedio;
};
const listaNumeros = [1, 2, 3, 4, 5];
const promedioNumeros = calcularPromedio(listaNumeros);
```

Ejercicio 3 Transformaciones

Nuestro cliente tiene un array de datos y nos a pedido que saquemos la siguiente información. El listado de los desarrolladores que tengan como habilidad “JavaScript” y el listado de los proyectos en el que sus desarrolladores trabajan.

Estos son los datos:

```
const datos = [
  {
    id: 1,
    nombre: 'Juan',
    habilidades: ['JavaScript', 'HTML', 'CSS'],
    proyectos: [
      { id: 1, nombre: 'Proyecto 1' },
      { id: 2, nombre: 'Proyecto 2' }
    ]
  },
  {
    id: 2,
    nombre: 'María',
    habilidades: ['Python', 'SQL', 'Django'],
    proyectos: [
      { id: 3, nombre: 'Proyecto 3' },
      { id: 4, nombre: 'Proyecto 4' }
    ]
  },
  {
    id: 3,
    nombre: 'Pedro',
    habilidades: ['Java', 'Spring', 'Hibernate'],
    proyectos: [
      { id: 5, nombre: 'Proyecto 5' },
      { id: 6, nombre: 'Proyecto 6' }
    ]
  }
];
```

Tenemos que hacer las operaciones necesarias para obtener estos 2 listados:

```
const desarrolladoresJavascript = [
  {
    "id": 1,
    "nombre": "Juan",
    "habilidades": ["JavaScript", "HTML", "CSS"],
    "proyectos": [
        { "id": 1, "nombre": "Proyecto 1"},
        { "id": 2, "nombre": "Proyecto 2" }
    ]
  }
]

const nombresProyectos = ['Proyecto 1', 'Proyecto 2', 'Proyecto 3',
'Proyecto 4', 'Proyecto 5', 'Proyecto 6']
```

Hay que crear un archivo **transform.js** con la solución.

Ejercicio 4 Arreglar bug de asincronia

Tenemos otro error que nuestro cliente nos pide arreglar. El cliente está pidiendo un usuario y nos dice que está usando el id correcto el 1. Pero que siempre le da undefined. Nos a pasado el código que tenemos que revisar y arreglar. Para este problema crear un archivo llamado **bugAsync.js** con la solución.

```
// Este programa simula una llamada asincrónica para obtener un usuario

function obtenerUsuario(id) {
  let usuario;

  setTimeout(() => {
    if (id === 1) {
      usuario = { id: 1, nombre: 'John Doe' };
    }
  }, 2000);

  return usuario;
}

const usuario = obtenerUsuario(1);
console.log(usuario);
```

Ejercicio 5: Catálogo Musical

Imagina que estás creando un sistema de gestión para un catálogo musical.

Cada canción tiene las siguientes propiedades:

Nombre de la Canción
Género
Duración (en minutos)

Implementa un programa que permita realizar las siguientes operaciones:

Agregar Canción: Permite al usuario ingresar información sobre una nueva canción y agrégala al catálogo.

Listar Canciones: Muestra en la consola la información detallada de todas las canciones en el catálogo. Si el catálogo está vacío, imprime un mensaje indicando que no hay canciones.

Buscar Canciones por Género: Pide al usuario que ingrese un género y muestra en la consola todas las canciones de ese género.

Calcular Promedio de Duración: Calcula y muestra en la consola el promedio de la duración de todas las canciones en el catálogo. (opcional)

Estructura Sugerida:

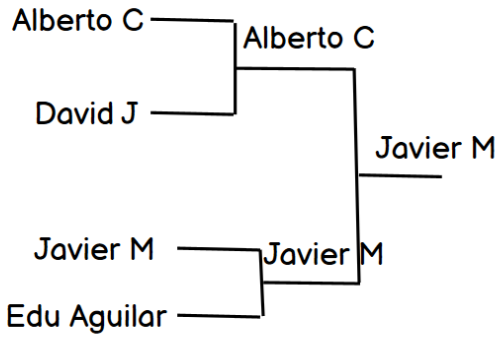
```
function crearCatalogo() {

  // ...

  return {

    agregarCancion: agregarCancion,

    listarCanciones: listarCanciones,

    buscarPorGenero: buscarPorGenero,

    calcularPromedioDuracion: calcularPromedioDuracion

  };

}

let miCatalogo = crearCatalogo();

// …
```

Proyecto Wimblecode

Nos acaba de contratar Wimblecode para que desarrollemos un software que registre los partidos y el marcador de cada encuentro. Para poder desarrollar este proyecto, nuestro cliente y propietario de Wimblecode nos ha proporcionado las reglas de este deporte para desarrolladores, de manera que podamos tener claro lo que tendremos que crear.

El torneo tiene un máximo y mínimo de 4 jugadores. “Alberto Casero”, “David Jiménez”, “Javier de Miguel”, “Eduardo Aguilar”.

Estos jugadores juegan un play-off, donde si ganan un partido con su rival avanzan a la siguiente ronda. Ejemplo de una partida:



Funcionamiento del torneo y el deporte Wimblecode.

- **¿Como se gana un torneo?** Si un jugador gana un partido, avanza y se enfrenta al siguiente jugador que ganó su partido. En la imagen se puede ver que el ganador del torneo fue Javier M.
- **¿Como se gana un partido?** Cada partido tiene juegos, el primero que gane 2 es el ganador del partido.
- **¿Como se gana un juego?** Para ganar un juego, el jugador tiene que cumplir estos requisitos:
    - Debe ganar 4 rondas.
    - Para que un juego se considere como victoria, debe tener una diferencia de 2 con respecto al otro jugador cuando llegue a 4. Es decir, si el jugador 1 tiene 4 rondas ganadas y el jugador 2 tiene 3, el jugador 1 aún no es considerado ganador. Tendría que ganar una quinta ronda.
    - El máximo de rondas es 7, en caso de un partido muy reñido.
- **¿Como se gana una ronda?** El sistema de puntuación es el siguiente:
    - Cada jugador puede tener alguno de estos puntos en un juego: 0, 15, 30, 40. Cada vez que un jugador se lleva un punto, la puntuación aumenta en este orden: 0 -> 15 -> 30 -> 40 -> Ganas*.
    - Si tienes 40 y ganas la siguiente tirada, ganas la ronda, pero hay reglas especiales:
        - Si ambos tienen 40 puntos, los jugadores están en "deuce" (empate).
        - Si el juego está en "deuce", el ganador de un punto obtendrá ventaja y si gana el siguiente punto ganaría la ronda.
        - Si el jugador con ventaja gana la pelota, gana la ronda.

- Si el jugador sin ventaja gana, vuelven a estar en "deuce".

Wimblecode se parece a otro deporte llamado tenis pero **no es igual, no confundir y seguir las reglas mencionadas anteriormente. Ya que hay diferencias.**

Funcionamiento del software a desarrollar

Nuestro cliente, nos dijo como debería de funcionar este software y nos dejo los métodos que quiere para poder probarlo. Este sería el ejemplo que nos dio:

Ejemplo de un partido:

```
// Ejemplo de software
const game = createMatch('Alberto C', 'David J');
// Cuando puntua el 1° judagor se registra de este modo
game.pointWonBy(1);
// Cuando puntua el 2° judagor se registra de este modo
game.pointWonBy(2);
// Quiero poder ver como va la ronda actual en todo momento
console.log(game.getCurrentRoundScore()); // Alberto C 15-15 David J
game.pointWonBy(1);
console.log(game.getCurrentRoundScore()); // Alberto C 30-15 David J
game.pointWonBy(2);
console.log(game.getCurrentRoundScore()); // Alberto C 30-30 David J
game.pointWonBy(1);
console.log(game.getCurrentRoundScore()); // Alberto C 40-30 David J
game.pointWonBy(2);
console.log(game.getCurrentRoundScore()); // Deuce
// jugador 1 toma ventaja
game.pointWonBy(1);
console.log(game.getCurrentRoundScore()); // Advantage Alberto C
// jugador 2 empata
game.pointWonBy(2);
console.log(game.getCurrentRoundScore()); // Deuce
// jugador 2 toma ventaja
game.pointWonBy(2);
console.log(game.getCurrentRoundScore()); // Advantage David J
// Con este punto jugador 2 gana la ronda
game.pointWonBy(2);
// Quiero poder ver como va la puntuación de un juego
console.log(game.getRoundsScore()); //  Alberto C 0 David J 1
// La primera ronda es para David le quedan 3 para ganar un juego
```

Con esto la puntuación queda de la siguiente manera

Player	Games	Rounds
Alberto	0	0
David	0	1

Si continuamos añadiendo puntos al jugador 2 por ejemplo.

```
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
// David gana 2° ronda

game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
// David gana 3° ronda

game.pointWonBy(2); // Player  2 wins the game
game.pointWonBy(2); // Player  2 wins the game
game.pointWonBy(2); // Player  2 wins the game
game.pointWonBy(2); // Player  2 wins the game
// David gana 4° ronda
// Primer juego ganado
console.log(game.getMatchScore()); // Alberto C 0 David J 1
```

Con esto la puntuación queda de la siguiente manera 0 juegos para Alberto y 1 juego para David. 0 rondas para alberto y 0 para david ya que comenzamos un juego nuevo.

Player	Games	Rounds
Alberto	0	0
David	1	0

Si continua ganando David, al final acabará ganando el partido.

```
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
// gana ronda 1°
game.pointWonBy(2); // Jugador 2 anota un punto
```

```
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
// gana ronda 2°
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
// gana ronda 3°
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
game.pointWonBy(2); // Jugador 2 anota un punto
// gana ronda 4°

// Método para ver los juegos de cada jugador
console.log(game.getMatchScore()); //  Alberto C 0\nDavid J 2
// método para ver el ganador. Si aún no hay ganador retornar null
console.log(game.getWinner()); // Output: "David J"
```

Player	Games	Rounds
Alberto	0	0
David	2	0

Con esto David ganaría el partido.

Para el torneo (Opcional)

Nuestro cliente quiere poder ver una simulación de que el software anterior funciona y nos pide simular un torneo antes de el usar los métodos anteriormente descritos.

Tenemos que crear una función que cree nuestro play-off de la captura anterior y que simule los partidos de forma aleatoria, hasta generar un ganador por partido y seguir avanzando en el torneo hasta el ganador final.

Como es una simulación y podemos añadir puntos a los jugadores de forma aleatoria. Para ello podemos usar

```
const randomPoint = Math.floor(Math.random() * 2) + 1;
```

Para obtener un valor random entre 1 y 2.

Para simular el torneo, recomiendo usar bucles, for y while.